TCP Congestion Window Controlling Algorithms on iSCSI Sequential Read Access

Machiko Toyoda †
machiko@ogl.is.ocha.ac.jp

Saneyasu Yamaguchi ‡
sane@tkl.iis.u-tokyo.ac.jp

Masato Oguchi †
oguchi@computer.org

† Ochanomizu University
Otsuka 2-1-1, Bunkyo-ku, Tokyo, Japan

‡ Institute of Industrial Science, The University of Tokyo
Komaba 4-6-1, Meguro-ku, Tokyo, Japan

Abstract

iSCSI is one of the most promising technologies in IP-SAN, which decreases a storage management cost. Congestion Window, one of TCP parameters, has a close relationship with system performance in accessing remote storage with iSCSI. In this paper, we propose a dynamic Congestion Window control method to stabilize the uneven behavior of throughput observed on the iSCSI storage access, and examine the algorithms to make the Congestion Window stabilized as soon as possible in the dynamic Congestion Window control method. We have adopted a linear search method, binary search method, and Congestion Window notification method. These algorithms are implemented on the experimental system and evaluated.

1 Introduction

As a large amount of data is processed in the network computing, Storage Area Network (SAN) has appeared and achieved extensive results. Both an introduction cost and a management cost of FC-SAN, based on Fibre Channel, are relatively high because FC products are expensive. Therefore, IP-SAN, configured with inexpensive Ethernet and TCP/IP, is expected as a next generation's SAN. iSCSI, represented as IP-SAN protocol, encapsulates SCSI command within a TCP/IP packet and transports it on a TCP/IP network. Consequently, server (Initiator) and storage (Target) can be connected as if the storage is attached to the server directly.

For improving performance of a remote storage access using iSCSI protocol, in this paper, we monitor the size of Congestion Window (CWND) inside Linux OS and propose an idea of a storage access with a dynamic CWND control method. Also, for accessing the storage using the proposed method effectively, we examine the control algorithms to stabilize CWND sooner. All algorithms are evaluated using the CWND control method in which remote storage is accessed.

The rest of this paper is organized as follows. Section 2 covers issues of accessing remote storage using iSCSI, and section 3 introduces the dynamic CWND control method. In section 4, we experiment using machines that implement the control algorithms and discuss the result, and section 5 presents the conclusion.

2 Issues of Accessing Remote Storage using iSCSI

iSCSI is ratified by the IETF in February 2003. In iSCSI, storage is called Target and server is called Initiator. iSCSI can use an existing TCP/IP network. Therefore, as broadband Internet is widely used, iSCSI is expected to be used for a remote storage access[1][2]. However, the performance of an iSCSI network is lower than that of a network based only on TCP/IP, due to its complex protocol processing overhead in accessing storage[3].

In order to examine the performance degradation of a remote storage access using iSCSI protocol, we have investigated the behavior of TCP layer, which is an important layer in TCP/IP communications. When Target is accessed from Initiator on a sequential read, it is observed that CWND shows various types of behavior depending on the conditions. That is to say, CWND increases and decreases repeatedly or remains stable, depending on the access block size[4]. Moreover, it is observed that the throughput decreases temporarily when CWND decreases, and the network performance is unstable in such a case.

In a long-latency network, which is supposed to be used for iSCSI, the data sender must wait for a long time until the data receiver sends back a reply packet of Acknowledgement (ACK) after it has sent the data packets. This waiting time increases as the number of packets sent at one time is fewer, and thus the throughput decreases dramatically. For this reason, the case of accessing remote storage in which CWND is stable is more efficient than the case in which the number of sent packets changes.

3 Dynamic Congestion Window Control Method

According to the relationship between CWND and throughput, which is described in the previous section, it is important for improving iSCSI performance that CWND is kept stable and the convergence time of it is shortened. For this reason, we propose the dynamic CWND control method to balance the unstable behavior of throughput.

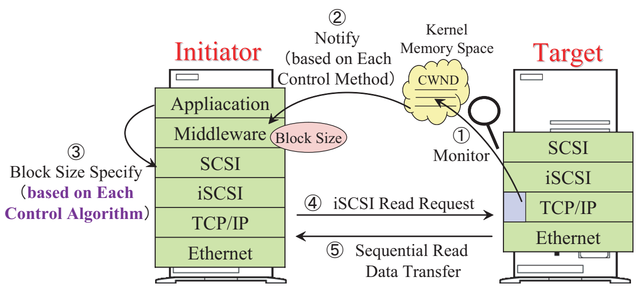

Figure 1. The concept of dynamic Congestion Window control method

Moreover, we examine a linear search algorithm, a binary search algorithm, and a CWND notification algorithm as control algorithms to stabilize CWND as soon as possible. In this section, we introduce the dynamic CWND control method based on each algorithm. Generally, user programs can't recognize the size of CWND because CWND is controlled in a Kernel space. Therefore we have inserted monitor functions in the TCP source code and implemented a recording mechanism of TCP parameters within a Kernel memory space, so that they are accessible from a User space. With this mechanism, we can observe TCP parameters by reading a special file for accessing the Kernel memory space. In the CWND control method, this mechanism is implemented on Target. An application of Initiator adjusts the block size of storage access by receiving a CWND notification from Target. CWND is reduced in the case that any of the following three errors happens; detecting Local Congestion error in which device driver buffer of the data sender overflows (CWR), receiving duplicated ACKs or SACK (Recovery), and detecting timeout (Loss). CWND is often reduced by the CWR error because in the storage access based on iSCSI protocol, the burst of packet transmission doesn't disappear easily. For this reason, we control CWND with the proposed method when CWND is reduced by the CWR error.
Linux TCP implementation doesn't change the window size unless the size of CWND, which has been set during the data transmission, is spent to zero before receiving ACKs. In this case, it may be possible to access remote storage with a longer block size than the current block size for a storage access. Therefore, the CWND control method tries making Initiator access remote storage with a longer block size than the current one if Target detects that CWND remains stable. The CWND control method is implemented as follows (Figure 1).

1. Target monitors CWND and observes its change.
2. Target notifies Initiator the information about CWND based on each control method, when CWR is detected so that CWND reduces, or CWND is stable.
3. When Initiator receives the notification, the middleware decides the block size based on each control algorithm and the application on Initiator modifies the block size.
4. Initiator sends a sequential read command for Target.
5. Target transmits the data of the requested block size.
6. This procedure iterates whenever Target detects CWR or decides that CWND is stable.

In the method based on the linear search algorithm, when Target requests the change of block size, it is modified at Initiator with the linear search algorithm. Target requests the decrease of block size when CWR is detected so that CWND reduces, and requests the increase of block size when CWND is stable. Also, Target judges the convergence of CWND by observing that CWND is stable after Target detects CWR error for the first time, or by observing that CWND is stable after Target detects CWR error so that CWND decreases in the case that Target judges CWND is stable at first.
The method based on the binary search algorithm is almost the same with that based on the linear search algorithm. The only difference is that the block size modified in Initiator is decided by the binary search algorithm. Target recodes the sent request for Initiator and the limit size of CWND (the maximum size of CWND without CWR error) for judging the convergence of CWND. The recorded CWND is compared with the current CWND when CWND is stable, and Target decides the convergence of CWND if the difference is sufficiently small.
In the method based on the CWND notification algorithm, Target notifies Initiator the size of CWND when CWR is detected so that CWND reduces, and notifies the limit size of CWND when CWND is stable. In addition, because Target recodes the size of notified CWND, Target can confirm the block size kept in Initiator. Therefore, Target is able to indicate the change of block size by perceiving the state of Initiator with this information. In this method, the specified block size calculated in the middleware is as follows.

Transmission Block Size [byte] = the size of CWND   Maximum Transmission Unit (MTU)

The size of MTU used in our experiment is 1448 Bytes excluding the TCP/IP header (including the option) from the maximum segment length of Ethernet (1500 Bytes). Target keeps the maximum block size (MaxBlockSize) calculated from the size of CWND in the case that CWND decreases, and the minimum block size (MinBlockSize) calculated from the size of CWND in the case that CWND is stable. The optimal block size with which CWND is stable exists between MaxBlockSize and MinBlockSize. Therefore, whenever Target detects that CWND is stable, the current block size calculated from CWND is compared with MaxBlockSize and MinBlockSize. Target judges by the difference whether CWND converges.
In each control method, it should be the optimal block size calculated from the proposed method when Target detects that CWND converges.

Table 1. Experimental Machines

CPU	Intel Xeon 2.4GHz
Main Memory	512MB DDR SDRAM
OS	Initiator, Target F Linux2.4.18-3 Dummynet F FreeBSD 4.9 - RELEASE
NIC	Initiaotr, Target F Intel PRO/1000XT Server Adapter Dummynet F Intel PRO/1000MT Server Adapter

4 Experiment of CWND Control Method using the Proposed Algorithms

In this section, we evaluate the performance of iSCSI sequential read access using the CWND control methods with an experimental system that implements each algorithm.

4.1 Experimental Setup

Our experimental system is as follows. We have established TCP/IP connection with Gigabit Ethernet between Initiator and Target. 1000Base-T Switching Hub is inserted in the case of an experiment with no-delay, and FreeBSD Dummynet[5] as an artificial delay machine is inserted in the case of an experiment with delay, between Initiator and Target, respectively. Initiator, Target, and Dummynet are constructed on Personal Computers. We have used Linux as an OS in Initiator and Target, and FreeBSD in Dummynet. In order to study the network performance on an iSCSI storage access, Target has been set up with the memory mode, in which a disk access isn't executed actually. Table 1 shows our experimental machines.
As an iSCSI implementation, we have used UNH IOL reference implementation ver.3 on iSCSI Draft 18[6] offered from the University of New Hampshire InterOperability Laboratory on the Target machine. In this UNH implementation, even if we issue a read command of a large block size, the SCSI layer divides the requested block into a small size. Thus the performance of storage access using iSCSI is degraded[3]. In our experiment, instead of using the UNH implementation on the Initiator machine, we have used an Initiator program written by ourselves. The program has equivalent functions to the UNH implementation of Initiator and can perform data transfer with a large block size. This Initiator program works as an application in the user space and communicates with Target using iSCSI protocol on a TCP/IP connection.

4.2 Outline of the Experiments

For evaluating the performance of iSCSI storage access in the implementation of the CWND control method using each algorithm, we have performed a sequential read access from Initiator to Target and have measured CWND, the block size, and throughput. The initial value of the block size to access Target at first is 128KB and 1024KB.

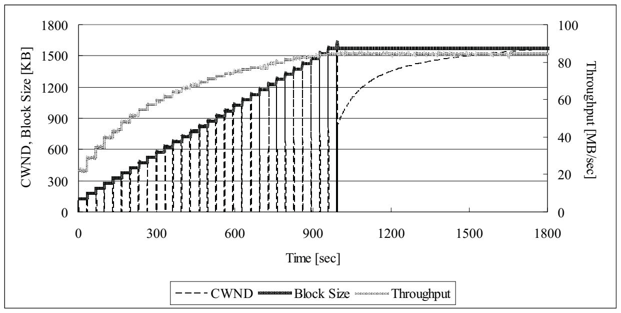

Figure 2. CWND, block size, and throughput using a linear search algorithm (block size: 128KB)

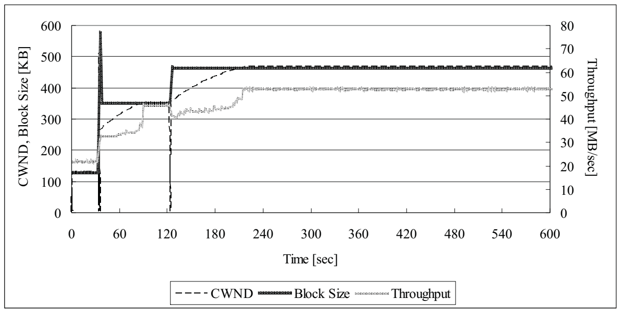

Figure 3. CWND, block size, and throughput using a binary search algorithm (block size: 128KB)

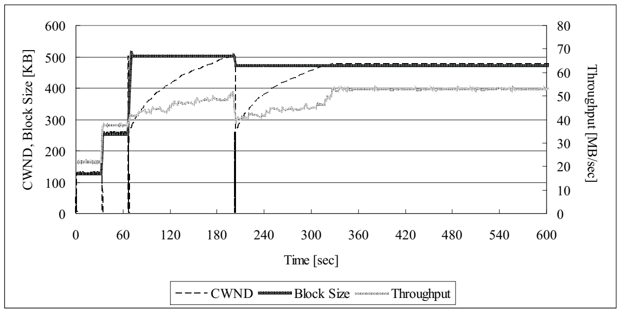

Figure 4. CWND, block size, and throughput using a CWND notification algorithm (block size: 128KB)

4.3 Experimental Result

Figure 2, 3, and 4 show the experimental result when the initial value of block size is 128KB. In this case, the buffer of a data sender doesn't overflow because the initial value of the block size is small. Therefore, because CWND is stable at the first storage access, Target issues an increase request of the block size and controls the storage access based on each algorithm. As a result, CWND is stabilized at the limit value.
Next, Figure 5, 6 and 7 show the experimental result when the initial value of block size is 1024KB. In this case, the buffer of a data sender overflows at first because the block size is too large. Therefore, Target issues a decrease request of the block size. After that, CWND is controlled based on

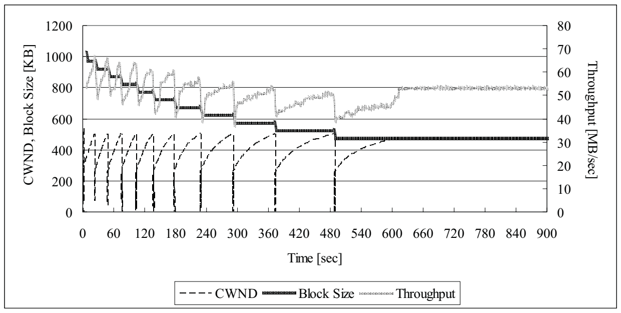

Figure 5. CWND, block size, and throughput using a linear search algorithm (block size: 1024KB)

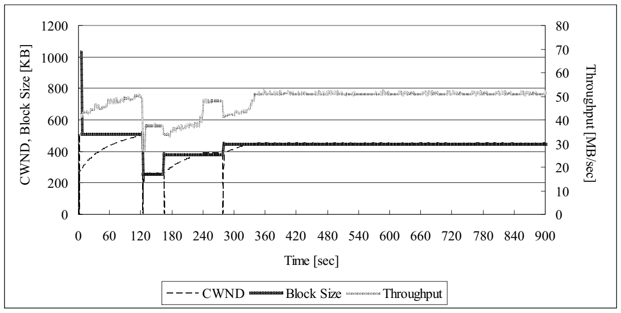

Figure 6. CWND, block size, and throughput using a binary search algorithm (block size: 1024KB)

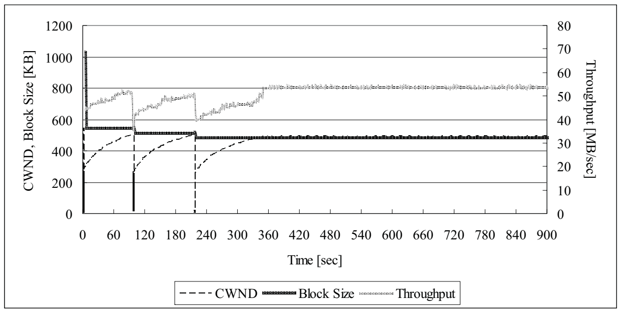

Figure 7. CWND, block size, and throughput using a CWND notification algorithm (block size: 1024KB)

each algorithm and converges at the optimal block size.

4.4 Discussion on the Result

In the method based on a linear search algorithm, it takes the longest time until CWND converges because Initiator accedes to the request from Target and simply repeats the increases or decreases of the block size. On the other hand, the method based on a binary search algorithm and a CWND notification algorithm converges sooner.
In the method based on a binary search algorithm, the access block size vibrates up and down due to the nature of the algorithm. In contrast, in the method based on a CWND notification algorithm, because the block size should be converged near from its current block size when CWR error detects, it doesn't change greatly. According to these reasons, we can conclude that CWND notification algorithm is the optimal algorithm in our proposed method.
In each algorithm, though CWND decreases suddenly when the increase request is issued, this is a temporary decrease and the period is about a few hundred micro seconds. Therefore, this hardly influences the performance.

5 Conclusion

In this paper, we have proposed the CWND control method to stabilize the uneven behavior of throughput observed on an iSCSI storage access. Moreover, the algorithms have been examined to make the Congestion Window settled to a stable value as soon as possible. Three algorithms have been adapted to our method. CWND, the block size and throughput have been measured with various initial value of the block size on an iSCSI sequential read access. As a result, the CWND notification algorithm is optimal among three algorithms.
As a part of future works, we will adapt the proposed method in a long-latency environment and evaluate its performance.

Acknowledgment

This project is partly supported by the Ministry of Education, Culture, Sports, Science and Technology, under Grant 13224014 of Grant-in-Aid for Scientific Research on Priority Areas.

References

[1] iSCSI Specification:
http://www.ietf.org/rfc/rfc3720.txt?number=3270/

[2] SCSI Specification:
http://www.danbbs.dk/˜dino/SCSI/

[3] S. Yamaguchi, M. Oguchi, and M. Kitsuregawa: ”iSCSI Analysis System and Performance Improvement of Sequential Access in Long-Latency Environment,” *IEICE Transaction on Information and Systems, Vol.J87-D-I, No.2*, pp.216-231, February 2004.

[4] M. Toyoda, S. Yamaguchi, and M. Oguchi: ”Relationship between TCP Congestion Window and System Performance on iSCSI Storage Access,” *Proc. 3rd Forum on Information Technology (FIT2004), B-004*, pp107-109, September 2004.

[5] L.Rizzo: ”dummynet”,
http://info.iet.unipi.it/˜luigi/ip_dummynet/

[6] InterOperability Lab: Univ, of New Hampshire,
http://www.iol.unh.edu/consortiums/iscsi/